OCR Error Correction Using a Noisy Channel Model

Okan Kolak
Computer Science and UMIACS
University of Maryland
College Park, MD 20742, USA
okan@umiacs.umd.edu

Philip Resnik
Linguistics and UMIACS
University of Maryland
College Park, MD 20742, USA
resnik@umiacs.umd.edu

ABSTRACT

In this paper, we take a pattern recognition approach to correcting errors in text generated from printed documents using optical character recognition (OCR). We apply a very general, theoretically optimal model to the problem of OCR word correction, introduce practical methods for parameter estimation, and evaluate performance on real data.

General Terms

OCR error modeling and correction

Keywords

OCR, noisy channel, parameter estimation, pattern recognition

1. INTRODUCTION

Optical character recognition (OCR) provides the means to make printed material available for on-line retrieval [16] and to extract valuable language data from hardcopy sources such as printed dictionaries [7]. Unfortunately, despite the claims of commercial vendors, OCR error rates are far from perfect, particularly as we move to more challenging languages such as Chinese and Arabic [8]. Moreover, almost all available systems are black boxes, and may not even permit customer-speci£c retraining. These constraints suggest an approach to OCR performance improvement based on post-processing of system output.

In this paper, we take a pattern recognition approach to correcting errors in OCR-generated text. We apply a very general, theoretically optimal model [14] to the problem of OCR string correction, provide practical methods for parameter estimation, and evaluate performance on real data.

2. OCR ERROR CORRECTION

Our pattern-recognition approach treats OCR as within the framework of the noisy channel model. Noisy channel approaches view a recognition problem generatively: the £rst step is to model the process by which an observed sequence of symbols $\mathcal{O}$ is generated from a true underlying sequence of symbols $\mathcal{T}$, and the second, at run time, is to identify the true sequence that is most likely given a particular observed sequence. More formally, given observed $\mathcal{O}$, one seeks to £nd the $\mathcal{T}$ that optimizes

$$\Pr(\mathcal{T}|\mathcal{O}) = \frac{\Pr(\mathcal{O}|\mathcal{T})\Pr(\mathcal{T})}{\Pr\mathcal{O}}$$

Since $\Pr(\mathcal{O})$ is constant across all possible $\mathcal{T}$, we can ignore it.

We focus on error correction at the word level: the problem is to identify the true word T given an observed word O in the output of an OCR system. In the approach presented here, we actually optimize $\Pr(O|T)$, which re¤ects two assumptions. First, we are assuming that there are no word-level merge or split errors (e.g. misrecognizing `forgive` as `for give`) — in the Appendix we give an algorithm for linking tokens in ground truth data with tokens in OCR output in the presence of missing/extra tokens.[1] Second, optimizing $\Pr(O|T)$ to £nd the best $\Pr(T|O)$ is justi£ed by assuming that $Pr(T)$ is uniform; we will take up better prior estimates for $\Pr(T)$ in future work.

Words O and T are presumed to be strings over a given alphabet of characters. Character-level splits and merges (e.g. recognizing `m` as `in` or `cl` as `d`) are approximated as combinations of insertion, deletion, and substitution errors.

2.1 OCR Error Correction as Syntactic Pattern Recognition

Oommen and Kashyap [14] introduce a general framework for syntactic pattern recognition that is well suited to the problem of OCR error correction. They provide a probabilistic generative model, M^*, that characterizes $P(O|T)$ for input and output strings T and O over a £nite alphabet A. They also give an ef£cient dynamic programming algorithm for calculating this probability of transforming one string to another.[2]

M^* is de£ned in terms of three probability distributions. The *quanti£ed insertion distribution*, G, controls insertions during the transformation process; in the most general case, it can be conditioned on the input string. The *quali£ed insertion distribution* de£nes the probability $Q(a)$, $a \in A$, that symbol `a` will be inserted, given that an insertion will be performed. The *substitution and deletion distribution* $S(b|a)$ governs the probability that symbol `a` in the input string will be transformed into symbol `b` in the output string; deletion is represented as a transformation into a special null symbol, λ. Let us de£ne and illustrate the generative process step by step, supposing the input string T is `abc`.

1. Randomly determine z, the number of insertions, using G.

[1] Doermann et al. [7] discuss learning-based approaches to zoning and segmentation.

[2] We have modi£ed their notation slightly for consistency with ours.

HLT 2002 San Diego, California, USA
.

Let us say z is 2 in this case.

2. Insert z instances of the special insertion symbol ξ into randomly selected positions in T, giving T'.

   Let us say we randomly selected positions 0 and 2, giving us ξabξc.

3. Randomly and independently substitute non-ξ symbols in T' using S to get T''.

   Suppose that a is substituted for a, λ substituted for b, and d is substituted for c. This gives us ξa$\lambda\xi$d. Recall that λ is the special null symbol that is used to represent deletion.

4. Randomly and independently transform all occurrences of ξ in T'' to symbols in A using Q, giving O'.

   Assume the £rst ξ is transformed into e, and the second one is transformed into f, giving eaλfd as O'.

5. Remove all occurrences of λ from O', giving O.

   We £nally get eafd, which is the output string O.

Oommen and Kashyap [14] prove M^* is stochastically consistent, optimal, and that it attains the information theoretic upper bound, and they do a simple study in the domain of correcting typographical errors using a simple uniform distribution for insertion and a confusion matrix for substitution. To our knowledge the work we are presenting here is the £rst published attempt at obtaining empirical parameter estimates for their model.

3. PARAMETER ESTIMATION

We assume that parameters of M^* are to be estimated from pairs $\langle t, o\rangle$, representing noisy channel input and output. In our setting t is the true word and o is the token recognized by the OCR algorithm, and maximum likelihood parameter estimation requires estimating event frequencies based on the word pairs. Since there are many ways of transforming t into o, it is necessary to distribute credit among all the possible sequences of events for the pair.

We investigated two methods for doing so. A simple £rst approach is to compute the lowest cost edit sequence based on edit distance, and assign all the weight to the events in that sequence. A second approach is to do iterative estimation over all sequences via an expectation-maximization (EM) approach [6].

3.1 Estimation Based on Levenshtein Distance

We base *edit distance estimation* (EDE) on picking the edit sequence for each $\langle t, o\rangle$ with the lowest Levenshtein edit distance [10]; this may not be optimal, but it is a likely sequence and ef£cient to compute. In our experiments, we assigned 0 cost to copy, and unit cost to insertion, deletion, and substitution events. Once the edit sequences are computed, calculating the event counts from edit sequences is quite simple. We smooth parameter estimations using Witten-Bell discounting [17].

3.2 Estimation based on Translation Models

It is interesting to note that our parameter estimation problem shares a great deal with the estimation problem for the IBM statistical machine translation models [2, 3]. For example, Figure 1 shows a word-level alignment representative of the IBM models and a character-level alignment consistent with M^*. Moreover, ef£cient parameter estimation algorithms for the IBM models have already been implemented in the GIZA++ software package [13].[3]

[3]For an introduction to the IBM translation models, see [9]

This suggests the possibility of using translation models as the basis for parameter estimation in the new setting of OCR error correction.

Figure 1: Examples of Alignment in Translation and OCR

The match between models is not perfect; for example, symbol re-ordering is crucial in translation but not OCR, and the IBM fertility model allows one-to-many symbol alignments (splits) but not many-to-one (merges). However, by treating characters as "words" in the translation model's vocabulary, it is possible to use unmodi£ed IBM-style model training on a parallel corpus of "sentence" pairs $\langle t, o\rangle$. The trained IBM model parameters can then be used to estimate the event counts to be used in maximum likelihood parameter estimation for G, Q, and S. We call this strategy *translation model estimation* (TME). EDE and TME are different in the way they estimate the event counts, but the rest of the computation is the same.

Estimation of the frequency of insertion counts is done using the Viterbi alignment between OCR and ground truth characters. All the OCR characters that are mapped from NULL, and all but one of multiple characters that are mapped from a single ground truth character are counted as insertions. For instance, if ground truth symbol a is mapped to symbols p and q, we can assume there is one insertion; however, we cannot tell which OCR symbol is the inserted one.[4] Therefore, we use the translation probability of NULL to a particular symbol as an approximation to that symbol's insertion probability. It is only an approximation, since it ignores the insertion cases that do not involve NULL. Speci£cally, we estimate the insertion count of a symbol s as

$$\begin{aligned} insertionCount(s) &= round(translationProb(NULL, s) \\ &\quad \times TotalInsertionCount) \end{aligned}$$

Note that we have three sets of insertion counts:

1. Total insertion count
2. Insertion count for each character
3. Insertion count frequencies

and they need to be consistent with each other. Informally, the total number of insertions implied by both insertion counts of symbols and frequencies of insertion counts should be consistent with the total insertion count. More formally:

$$\begin{aligned} TotalInsertionCount &= \sum_{i>0} insertionCountFreq(i) \times i \\ &= \sum_{s \in \{symbols\}} insertionCount(s) \end{aligned}$$

where $insertionCountFreq(i)$ is the number of edit sequences with i insertions.

We estimate the total number of insertions and insertion count frequencies together, so that they are consistent. We then round insertion counts for some symbols up or down to make their sum consistent with the total insertion count.

[4]We are ignoring less likely cases such as one deletion with two insertions.

For substitution counts, we will use the translation probabilities. However, deletion is considered a special case of substitution, and there is no symbol-to-NULL translation probability; therefore, we use the 0-fertility probability of a symbol to estimate its deletion count. If a symbol s appears n times in the corpus, then its deletion count is estimated as

$$deletionCount(s) = round(symbolCount(s) \times fertilityProb(s, 0))$$

Once having accounted for deleted symbols, we can distribute the remaining counts to substitutions. This means that substitution count of a symbols s with a symbol t is estimated as

$$substitutionCount(s,t) = round((symbolCount(s) - deletionCount(s)) \times translationProb(s,t))$$

Note that for any symbol, the sum of deletion and substitution counts should be equal to the symbol count for that symbol.

$$symbolCount(s) = deletionCount(s) + \sum_{t \in \{symbols\}} substitutionCount(s,t)$$

This consistency is trivially achieved by setting the symbol count to the number imposed by deletion and substitution counts. Also note that we ignore the consistency of our estimated counts and actual counts in the data for simplicity, since it is not crucial for validity of our estimates.

4. EVALUATION

We performed several sorts of evaluations of our approach. First, as a preliminary evaluation on artificial data, we performed a "reality check" experiment very similar to Oommen and Kashyap's [14] study. The goal here was to see how well the parameter estimation techniques could recover a set of known parameters by training on a data set generated artificially using those parameters. Our results were very similar to Oommen and Kashyap's. We created our training data set by using M^* with a fixed set of parameters similar to theirs to generate a noisy version of 30158 common English words with lengths between 7 and 14 characters. We then estimated the values of the parameters using our two training methods, varying the quantity of training data. The Kullback-Leibler (K-L) distance between the parameter estimates and the true parameters that generated the data are given in Figure 2 as a function of the quantity of training data. There was no significant difference between EDE and TME training methods; both methods perform quite well when the M^* model is known to be a good characterization of the data.

In our second evaluation scenario, largely because the IBM translation modeling framework made it an easy experiment to perform, we looked at whether or not a straightforward statistical MT decoding approach would succeed in recovering an original ground truth string from its OCRed counterpart. Treating characters as "words" in the vocabulary, we used the CMU-Cambridge Toolkit [5] for language modeling, GIZA++ for translation modeling, and the ISI ReWrite Decoder [12] for decoding. This approach performed very poorly, with decoder output on a test set rarely producing the ground truth word, possibly because it does not require that the decoded sequence of characters actually form a word. Modifying the language model component could solve the problem, but for the time being we are not pursuing this route.

Instead, our third evaluation scenario was to take an incorrectly recognized word w and to select the correct word d from a dictionary D by ranking the choices according to $\Pr_{M^*}(w|d)$.[5] We present accuracy figures for selecting the correct d as the top choice (1-best), and as one of the top 3 choices (3-best).

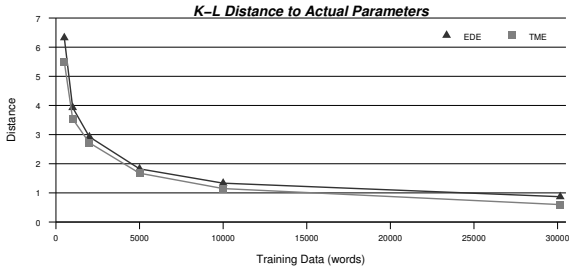

Figure 2: K-L Distance between Original and Learned Parameters

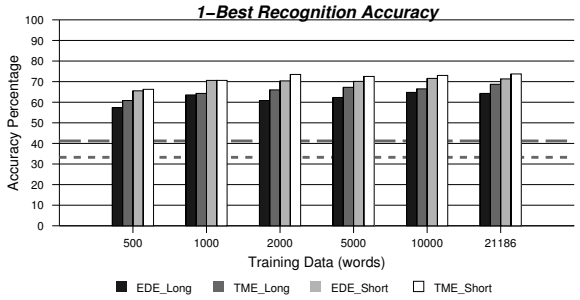

Figure 3: Accuracy of 1-Best Recognition for OCR Data

For artificial training and test data, where noisy English strings are generated from ground truth English words as described in the first evaluation, above, the 1-best recognition accuracy was around 96%. This confirms that both training methods perform quite well in an artificial scenario where the generative process is known to be an instance of the M^* model.

More realistically, we also experimented on actual OCR data generated by scanning a hardcopy of the book of Genesis in French using a commercial OCR package, OmniPage 8.0 (which supports French). Measuring against ground truth, the word error rate on these data is approximately 20%. We used disjoint training and test sets, varying the size of the training data to explore sensitivity to training set size; the test set included 423 misrecognized words. We experimented with a short dictionary (1005 words) and a long dictionary (4300 words) in order to explore sensitivity to the number of alternatives for d. The number of valid-word errors was 8 for the short dictionary and 17 for the long dictionary, and these are not included in accuracy computations. The results in Figure 3 and Figure 4 respectively show the percentage of the time that the correct word d is the first choice, and the percentage of the time that it occurs among the top 3 choices. Actual word counts are given in Table 1 and Table 2.

Dashed horizontal lines in the figures are baseline performance of picking d as the word with the lowest Levenshtein edit distance. Baseline 1-best recognition accuracy is 41.20% for the long dictionary and 33.25% for the long one; for the 3-best case, the numbers

[5]Incorrect tokens are identified because they are not in the dictionary. As in many approaches to spelling correction, we are not considering valid-word errors here; see Section 5.

Training Size	Long Dictionary		Short Dictionary	
	EDE	TME	EDE	TME
500	233	247	272	275
1000	258	261	293	293
2000	247	268	292	305
5000	253	273	291	301
10000	263	270	297	303
21186	261	279	296	306

Table 1: 1-Best Recognition Count for OCR Data

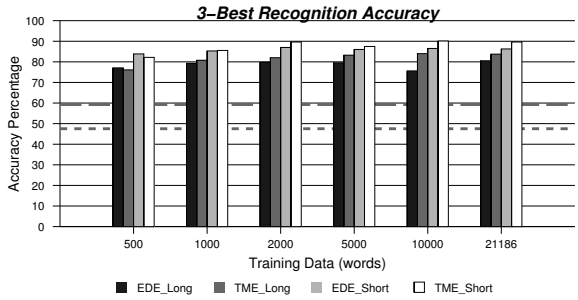

Figure 4: Accuracy of 3-Best Recognition for OCR Data

Training Size	Long Dictionary		Short Dictionary	
	EDE	TME	EDE	TME
500	313	309	348	341
1000	322	328	354	355
2000	324	333	361	372
5000	323	338	357	363
10000	307	341	359	374
21186	327	340	358	372

Table 2: 3-Best Recognition Count for OCR Data

are 59.27% and 47.53%, respectively. The difference between our methods and the baseline is clear: we nearly double the baseline performance one would obtain in correcting OCR word errors by picking the word in the dictionary with the lowest Levenshtein edit distance. In addition, TME training consistently performs a bit better than edit distance training (The difference is statistically significant when aggregated across all experimental conditions, but not condition-by-condition). This is most likely because EDE ignores all but one of the possible edit sequences that could transform an original word to the noisy one, and it may also re¤ect the fact that the IBM translation models are able to capture the split errors in OCR.

The £gures suggest that the size of the candidate dictionary has a signi£cant effect on accuracy, hence one can expect to improve accuracy by using a clever candidate selection mechanism or incorporating evidence from a language model. We were interested to observe that the results do not improve much with the size of the training data amount: training with 2000 words produces reasonable results, which makes practical the creation of manually generated ground truth for training.

5. RELATED WORK

We are unaware of published work on post-recognition error correction for OCR generated text, nor of other work that combines the theoretical guarantees of M^* with the practical aspects of our estimation methods. The spelling correction literature provides methods which could be applied to the OCR correction problem, however, so we brie¤y compare our method with some spelling correction methods.

Church and Gale [4] used probability scores for spelling correction for the £rst time. However, their work is limited to the correction of words with a single spelling error and cannot be used to correct OCR text that may contain multiple errors per word.

Mangu and Brill [11] proposed a rule-based approach for spelling correction. Unlike models based on character edit operations, their method is capable of correcting valid-word errors. Moreover, the output of the model is understandable by humans. However, it is based on word usage errors and requires pre-speci£ed confusion sets, which severely limits its utility for OCR correction.

Ristad and Yianilos [15] proposed a stochastic edit distance model and methods to learn model parameters from a training corpus. They achieved a much lower error rate for a word pronunciation problem compared to non-probabilistic edit distance. Our method is very similar; however, their model restricts the possible probability distributions, whereas the model we use permits arbitrarily distributed syntactic errors.

Brill and Moore (B&M) [1] introduced an improved noisy channel model for spelling correction that allows arbitrary string-to-string edit operations. They get better error correction results compared to methods that use only character edits. The authors kindly trained their system on two of our training corpora (with 2000 and 21186 words) and tested on our test data.[6] Their method, unlike ours, only trains on pairs containing recognition errors; there were 371 such pairs in the small (2000-pair) set and 3826 in the large (21186-pair) set. The comparison between TME and their method is given in Table 3. B&M performs signi£cantly better for the larger training set, though the difference in performance for the smaller training set is not statistically signi£cant.

We performed a more detailed analysis on the performance of

[6]Their spell checker had only been developed for and debugged/tested on English, and was used on our French data without modi£cation.

Training Set Size	1-Best		3-Best	
	B&M	TME	B&M	TME
371/2000	281	268	333	333
3826/21186	327	279	365	340

Table 3: Recognition Counts for Brill & Moore vs. TME

EDE, TME, and B&M, for the large training set and 3-Best recognition. We found that 26 of the test words could not be recognized by any of the methods, 310 were recognized by all, and 25 were recognized by two of the three. Each cell in Table 4 shows how many words recognized by one method (rows) were not recognized by the other methods (columns); for example, TME was correct on 12 words that B&M missed. The fact that all three methods are recognizing some words not recognized by other methods means a combination of the three might perform better. Although a straightforward voting scheme is not enough (only 325 words get a majority vote, lower than individual scores), some other combination method may prove useful.

	EDE	TME	B&M
EDE	0	6	10
TME	20	0	12
B&M	52	40	0

Table 4: Pairwise Exclusive Word Recognition Comparison

6. FUTURE WORK

We currently use uniform prior probabilities for the correction candidates. Using a scoring and/or elimination method for candidates, e.g., based on an n-gram language model for words, would almost certainly improve performance. Such potential scoring methods and their effect on correction performance requires further work.

For some languages, necessary resources, including an appropriate online dictionary, may not be available. Creating correction candidates with limited resources, possibly using the OCR documents being corrected, is very appealing and deserves further investigation.

7. CONCLUSIONS

We have presented an approach to string error correction that is based on a model with attractive theoretical properties, contributing new techniques for empirical parameter estimation and investigating application to OCR errors. The approach requires a practical quantity of manually generated training data, is ef£ciently computed, and works signi£cantly better than the na˜ve approach of correcting errors by £nding the most similar dictionary entry according to simple edit distance. We observed that the size of the candidate dictionary signi£cantly affects correction accuracy, and, since the model provides well de£ned probabilities, the door is open to integration with language model predictions or other stochastic criteria for ranking likely correction candidates.

Acknowledgments

This research was supported in part by DARPA/ITO Cooperative Agreement N660010028910 and Department of Defense contract MDA90496C1250.

We are grateful to F. Och and H. Ney for GIZA++, D. Marcu for the ISI decoder, P. Clarkson and R. Rosenfeld for the CMU-Cambridge toolkit, E. Brill for his kind assistance in comparative evaluation, and B. Byrne and B. Oommen for useful discussion.

8. REFERENCES

[1] E. Brill and R. C. Moore. An improved model for noisy channel spelling correction. In *38th Annual Meeting of the Association for Computational Linguistics*, pages 286–293, Hong Kong, China, October 2000.

[2] P. F. Brown, J. Cocke, S. A. D. Pietra, V. J. D. Pietra, F. Jelinek, J. D. Lafferty, R. L. Mercer, and P. S. Roossin. A statistical approach to machine translation. *Computational Linguistics*, 16(2):79–85, 1990.

[3] P. F. Brown, S. A. D. Pietra, V. J. D. Pietra, and R. L. Mercer. The mathematics of machine translation: Parameter estimation. *Computational Linguistics*, 19(2), 1993.

[4] K. Church and W. Gale. Probability scoring for spelling correction. *Statistics and Computing*, (1):93–103, 1991.

[5] P. Clarkson and R. Rosenfeld. Statistical language modeling using the CMU-Cambridge Toolkit. In *ESCA Eurospeech*, 1997.

[6] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[7] D. Doermann, H. Ma, B. Karagöl-Ayan, and D. W.Oard. Translation lexicon acquisition from bilingual dictionaries. In *Ninth SPIE Symposium on Document Recognition and Retrieval*, San Jose, CA, 2002. to appear.

[8] T. Kanungo, P. Resnik, S. Mao, D. wan Kim, and Q. Zheng. The Bible, truth, and multilingual optical character recognition. submitted for publication.

[9] K. Knight. A statistical MT tutorial workbook, April 1993. http://www.isi.edu/natural-language/mt/wkbk.rtf.

[10] V. I. Levenshtein. Binary codes capable of correcting deletions, insertions and reversals. *Soviet Physics Doklady.*, 10(8):707–710, 1966.

[11] L. Mangu and E. Brill. Automatic rule acquisition for spelling correction. In *14th International Conference on Machine Learning*, pages 187–194. Morgan Kaufmann, 1997.

[12] D. Marcu and U. Germann. The ISI ReWrite Decoderrelease 0.7.0b. http://www.isi.edu/˜germann/software/ReWrite-Decoder/.

[13] F. J. Och and H. Ney. Improved statistical alignment models. In *ACL00*, pages 440–447, Hong Kong, China, October 2000.

[14] B. Oommen and R. Kashyap. A formal theory for optimal and information theoretic syntactic pattern recognition. *Pattern Recognition*, 31:1159–1177, 1998.

[15] E. S. Ristad and P. N. Yianilos. Learning string edit distance. *IEEE Transactions on Pattern Recognition and Machine Intelligence*, May 1998.

[16] Y.-H. Tseng and D. W. Oard. Document image retrieval techniques for chinese. In *Fourth Symposium on Document Image Understanding Technology*, pages 151–158, Columbia, MD, April 2001.

[17] I. H. Witten and T. C. Bell. The zero-frequency problem: Estimating the probabilities of novel events in adaptive text compression. *IEEE Transactions on Information Theory*, 37(4):1085–1093, July 1991.

APPENDIX

Word Pairing with Missing Word and Extra Word Errors

The parameter estimation methods we propose require word pairs $\langle t, o \rangle$ for training. Automatically extracting these word pairs from OCR/ground-truth document pairs is not a trivial task owing to missing and extra word errors. These errors can be caused by merging or splitting words as a result of whitespace recognition errors, missing words, and misrecognizing stains and marks as words. The possibility of having missing or extra words renders useless any simple position-based pairing between OCR tokens and ground truth tokens. Therefore, we designed a simple algorithm to compute a likely pairing. The algorithm assumes that many words are recognized correctly for each line in the OCRed document, and uses these words as anchor points for pairing.

Correctly recognized words are identified by finding a sequence of word copy, insertion, deletion, and substitutions operations that converts ground truth into OCR version with the minimum cost. Copied words in this edit sequence are assumed to be correctly recognized (and momentarily we will generalize this to non-identical but similar words). Owing to the space requirements of the edit distance algorithm, we assume OCR and ground truth files are aligned at the line level, and work on one line pair at a time.

Let us present the algorithm using a sample execution where the original (ground truth) line is `a b c d e` and the OCRed line is `p b q e r`.

- Find an edit sequence that has the minimum cost. In case of equality, the sequence with more copy operations is favored.

```
Substitution : a --> p
Copy         : b --> b
Deletion     : c -->
Substitution : d --> q
Copy         : e --> e
Insertion    :   --> r
```

- For each copy operation, create a bracketed component on both sides that contains only the copied word.

```
a [b] c d [e]
p [b] q [e] r
```

- Do the following for both the original and noisy lines:
  - For each existing bracketed component, create a new bracketed component that contains everything to its right up to the next bracket, or up to the end of the line if there is no next bracket.

```
a [b] [c d] [e] []
p [b] [q] [e] [r]
```

  - Create a bracketed component that contains everything from the beginning of the line to the first existing bracket, or to the end of the line if there is none.

```
[a] [b] [c d] [e] []
[p] [b] [q] [e] [r]
```

- For each bracketed component in the original line, pair its contents with the corresponding (left-to-right) component in the noisy line, ignoring empty bracket pairs.

```
'a'     -->     'p'
'b'     -->     'b'
'c d'   -->     'q'
'e'     -->     'e'
NULL    -->     'r'
```

Since pairing is based on the edit sequence generated by the edit distance algorithm, one can tweak it by modifying the edit distance algorithm or its parameters.

One such useful modification is to use a similarity threshold instead of exact match to decide if two words match. To give an example, let `to illustrate this` be the original line and `lo ilustrate ths` be the OCR line. With exact matching, we would get the following edit sequence

```
Substitution : to         --> lo
Substitution : illustrate --> ilustrate
Substitution : this       --> ths
```

and the pairing `to illustrate this` → `lo ilustrate ths`, where the whole string is mapped over as a single chunk. However, if we assume that two words match if their edit distance is less than 20% of the length of the longer one, the edit sequence would become

```
Substitution : to         --> lo
Copy         : illustrate --> ilustrate
Substitution : this       --> ths
```

giving the more desirable pairing `to` → `lo`, `illustrate` → `ilustrate`, and `this` → `ths`.

Another useful modification is to change the cost of the copy operation. Assume the original line is `a b c d e` and the OCR line is `e p q r s`. With 0 copy cost, we would get the following edit sequence:

```
Substitution : a --> e
Substitution : b --> p
Substitution : c --> q
Substitution : d --> r
Substitution : e --> s
```

and the corresponding pairing would be `a b c d e` → `e p q r s`. If we use -4 as the copy cost, the edit sequence would become

```
Deletion     : a -->
Deletion     : b -->
Deletion     : c -->
Deletion     : d -->
Copy         : e --> e
Insertion    :   --> p
Insertion    :   --> q
Insertion    :   --> r
Insertion    :   --> s
```

and the pairing would be `a b c d` → `NULL`, `e` → `e`, and `NULL` → `p q r s`. By changing the cost of the operations, one can tune how far an original word and its correctly recognized version on the noisy side can be.